Report di progetto su Neural Networks

Segmentazione Semantica e implementazione di una applicazione per la creazione di sticker

Andrea Ceccato, Luca Dolci, Lorenzo Farinea

Programmazione di Sistemi Embedded, Luglio 2019

Indice

1 NNAPI e TensorFlow Lite

In questa sezione si illustreranno due interfacce per risolvere problemi di Intelligenza Artificiale sulla piattaforma Android.

1.1 NNAPI

A partire dalla versione 8.1 (API level 27) di Android è stata introdotta la *Android Neural Network API* (NNAPI). Si tratta di un' interfaccia di basso livello nata con lo scopo di portare l' esecuzione di modelli machine learning direttamente sui dispositivi mobili. Questo tipo di elaborazione sta permettendo grandi passi in avanti in svariati campi applicativi quali ad esempio image classification, object detection, semantic segmentation, pose estimation, gesture recognition, smart reply, speech recognition.

Tipicamente l'interazione tra questa tecnologia e i dispositivi mobili avviene solo in maniera indiretta: le informazioni di input che vengono prese dall' utente ed inviate attraverso la rete per poter essere poi elaborate in una *server farm*, la quale in seguito invia il risultato dell' inferenza. Grazie alla NNAPI invece tutto il processo, dalla raccolta dei dati fino alla produzione dell' output, avviene sul dispositivo stesso portando così a notevoli vantaggi in termini di:

- responsiveness: in applicazioni critiche l' utente risente molto meno della latenza dovuta al calcolo;
- disponibilità: la rete funziona anche in assenza di connessione;
- velocità: l' uso di progessori appositi quali NPU *Neural Process Unit*) e GPU velocizzano il tempo di inferenza;
- sicurezza: i dati dell'utente non vengono inviati tramite internet;
- costo: non serve possedere una server farm.

Ovviamente svolgere tutta l'elaborazione sul dispositivo porta anche a degli svantaggi tra i quali citiamo l' utilizzo del processore per un'unità di tempo maggiore con conseguente consumo della batteria oltre alla dimensione dell' applicazione a causa del modello che senza alcuni accorgimenti potrebbe avere dimensioni considerevoli.
Come accennato precedentemente NNAPI di Android è un' interfaccia di basso livello scritta in C++ che si relaziona direttamente con i driver dell' hardware disponibile sul dispositivo mobile potendo così distribuire, a runtime, il carico di lavoro computazionale tra i processori disponibili sulla macchina stessa.

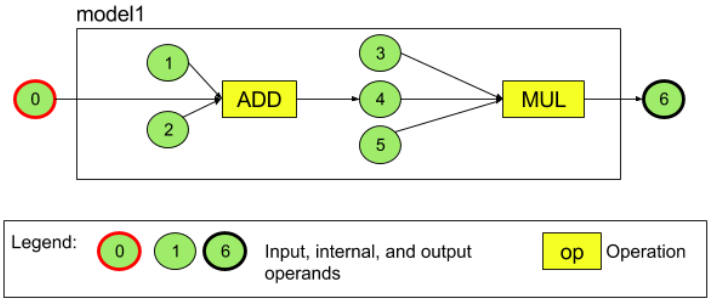

Figura 1: Esempio di un modello

Dall' altro lato NNAPI è nata per fornire un livello di funzionalità base per framework di machine learning di più alto livello tra i quali TensorFlow Lite, con i quali viene costruita ed allenata la rete neurale.

Per poter eseguire computazione usando NNAPI è necessario aver costruito un grafo diretto detto grafo di computazione. Tale grafo è costituito da operazioni matematiche e da valori costanti che sono stati "imparati" durante la fase di *training*. Le operazioni matematiche tipicamente utilizzate in questo contesto sono addizione, moltiplicazione, convoluzione e varie funzioni di attivazione tra cui Sigmoid e ReLU. Le costanti invece sono contenute nei cosiddetti "operandi" che possono essere scalari (singoli numeri 32-bit floating point, integer o unsigned integer) oppure tensori (array multidimensionali). In figura 1 viene mostrato un esempio di modello in cui vi sono sette operandi e due operazioni.

1.2 TensorFlow Lite

TensorFlow Lite nasce come estensione della libreria *TensorFlow* sviluppata da Google con lo scopo di fornire un insieme di strumenti per poter eseguire i modelli di reti neurali su dispositivi mobili e IoT. *TensorFlow Lite* è formato da due principali componenti il *TensorFlow Lite Interpreter* e il *TensorFlow Lite Converter*.
Il convertitore di TF Lite viene usato per convertire i modelli di TensorFlow in un formato FlatBuffer, in modo tale che possano essere utilizzati dall'interprete. FlatBuffer è una libreria di serializzazione multipiattaforma creata da Google per molti dei principali linguaggi di programmazione. I suoi principali vantaggi sono la facilità di accesso ai dati del buffer binario, l'efficienza in termini di memoria e velocità nonché la flessibilità e la compatibilità con svariate piattaforme.
Il convertitore dunque genera un file FlatBuffer di TensorFlow Lite (.tflite) a partire da un modello di TensorFlow. La conversione avviene tipicamente tramite delle API in Python e supporta modelli di vario tipo ("SavedModels", "frozen GraphDef", "HDF5"). In figura 2 è mostrato il workflow completo.

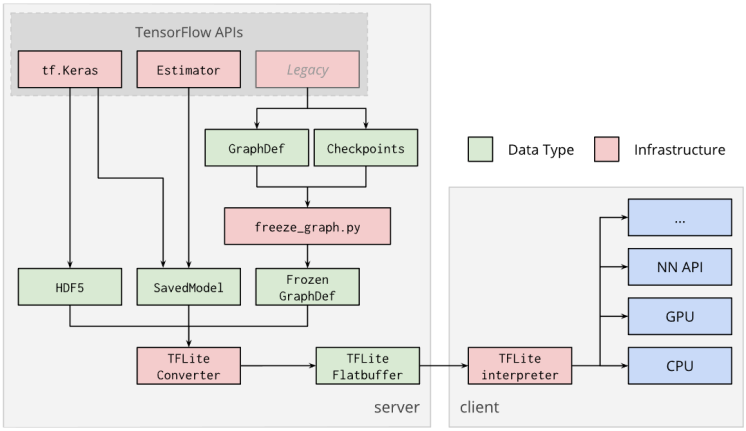

Figura 2: Workflow di TensorFlow Lite

Una volta ottenuto il modello convertito a TensorFlow Lite viene eseguita l'inferenza sul dispositivo mobile attraverso l'interprete. Quest'ultimo, di dimensione binaria ridotta ottimizzata per l'utilizzo in Android, iOS, sistemi embedded Linux e microcontrollori, supporta una serie di operatori di base per poter eseguire inferenza.

1.2.1 Quantizzazione

Esistono alcune tecniche per l' ottimizzazione della rete neurale post-training tra le quali la quantizzazione. Questa operazione permette di ridurre la dimensione del modello migliorando così il tempo di inferenza su CPU e accelerazione hardware con una minima perdita in merito all' accuratezza. La quantizzazione consiste nella conversione di tutti gli operatori e i pesi della rete da floating point a 8-bit di precisione. É importante controllare la perdita di precisione con degli strumenti di valutazione delle prestazioni per far si che si mantenga all'interno dei limiti di accettabilità.

1.2.2 Possibilità di sfruttare la GPU

I processori grafici sono disegnati per aver un grande efficienza nell' esecuzione di carichi di lavoro estremamente parallelizzati. Queste caratteristiche li rendono particolarmente adatti alle reti neurali che consistono in un enorme quantità di operatori, ognuno dei quali lavora con dei tensori che possono facilmente essere suddivisi in carichi di lavoro più piccoli e portati avanti in parallelo. L' impiego delle GPU in questo campo ha ridotto notevolmente il tempo di inferenza permettendo lo sviluppo di applicazioni *real-time* precedentemente impensabili. Le GPU infatti lavorano con numeri in virgola mobile a 16 o 32 bit. Inoltre, eseguono la computazione efficientemente, consumano meno potenza e scaldano di meno rispetto a una CPU.

2 Esempio di utilizzo di TensorFlow Lite: segmentazione semantica

Uno dei tanti esempi di utilizzo di una rete neurale nell' ecosistema Android è la *segmentazione semantica*. Questa, basandosi sulle reti neurali convoluzionali, si prefigge di riconoscere ed evidenziare figure di oggetti all' interno di una immagine, assegnando una etichetta per ogni pixel che la compone.

2.1 Le reti neurali convoluzionali

In molti campi di riconoscimento di oggetti o forme in immagini, le reti neurali convoluzionali sono spesso le più utilizzate, la loro struttura infatti si ispira infatti alla disposizione dei neuroni nella corteccia visiva negli animali, permettendogli di svolgere efficacemente questo genere di elaborazioni.
La gerarchia dei neuroni parte con un primo livello che associa un neurone al relativo pixel, seguono diversi livelli nei quali le connessioni sono locali, per ridurre drasticamente il numero di connessioni, e i pesi comuni, ovvero condivisi a gruppi. Il livello di output è invece completamente connesso al livello precedente. Le località delle connessioni fa sì che parti diverse della immagine vengano processate nello stesso modo, ciò fa si che si possa parlare di covoluzione tra un livello e il successivo, filtrando gruppi di pixel applicando una maschera di pesi sempre uguale per ogni gruppo (grazie ai pesi condivisi).
I neuroni di ogni livello possono essere rappesentati per volumi: ad ogni livello viene dunque applicato un filtro 3D, che può essere 'spostato' di un certo fattore *stride*, generalmente è diverso dall' unità, riducendo il volume del livello successivo in altezza e in larghezza. La lunghezza viene rappresentata come *feature map*, dove per ogni 'fetta' di volume si ha la condivisione dei pesi, aumenta ad ogni filtraggio. E' possibile incastrare dei livelli di *pooling* per ridurre le dimensione delle feature map, aggregando le informazioni (*subsampling*).

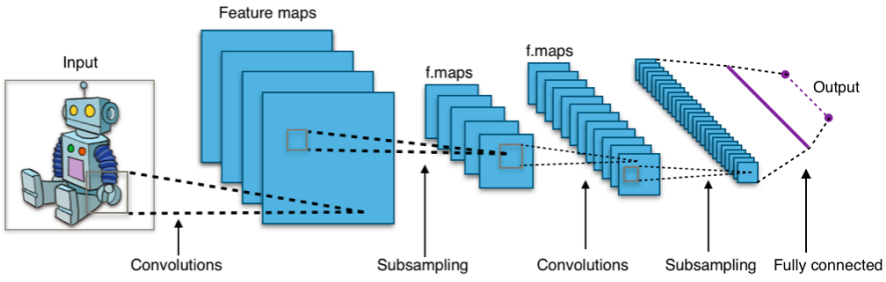

Figura 3: Classica architettura di una CNN. Fonte: creazione originale di Aphex34 per Wikimedia Common

2.2 Segmentazione semantica con DeepLab

Una maniera di effettuare segmentazione semantica all' interno di TensorFlow è quella di affidarsi a DeepLab [1], una recente tecnologia sviluppata da Google e implementata nei software delle fotocamere all' interno dei loro recenti smartphone [2].
La rete, arrivata alla terza versione (DeepLab-v3+), si basa su potenti reti neurali convoluzionali, le principali sono *MobileNetv2* e *Xception*. Mentre la seconda è utilizzata in ambiti server per avere una estrema accuratezza, la prima è utilizzabile anche in ambiti Mobile e IoT, con risultati più che soddisfacenti. É possibile allenare la rete su diversi datasets: *Cityscapes* per segmentare elmenti architettonoci di città e *PASCAL VOC2012* e *ADE20K*, più orientati ad oggetti generici.
L' architettura della rete si basa su diverse tecniche applicate alla struttura base di una CNN:

- **Atrous Convolution**: il 'filtraggio' di parte della feature map non viene eseguito considerando valori di input adiacenti ma, definito un passo, la maschera di filtro viene estesa, pesando solo gli input ai vertici del filtro e nel punto medio tra questi. In figura 4 vi è un esempio di questo tipo di convoluzione.

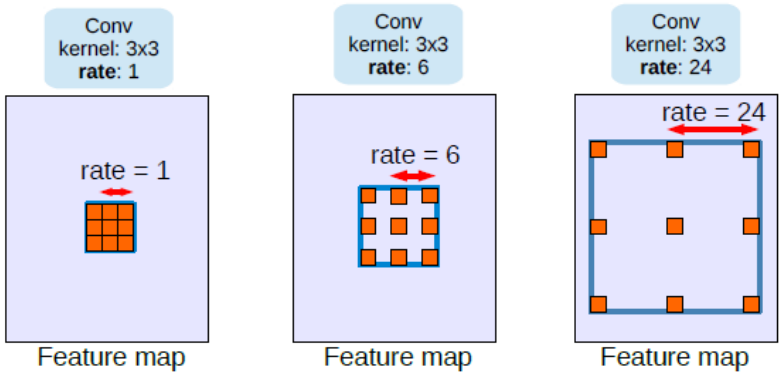

Figura 4: Atrous Convolution. Fonte: *towardsdatascience.com*

In simboli questo tipo di convoluzione è la seguente: $\boldsymbol{y}[i] = \sum_k \boldsymbol{x}[i + r \cdot k]\boldsymbol{w}[k]$, con $\boldsymbol{y}$, $\boldsymbol{x}$, $\boldsymbol{w}$ rispettivamente output, input e peso. i è il punto dove calcolare l' output, k l' indice di iterazione sul kernel e il parametro r è il *rate*, che corrisponde al passo usato per 'allargare' il filtro. Questo parametro è fondamentale per definire un trade-off tra accuratezza locale (per esempio per riconoscere un bordo in una piccola area di

[1] https://github.com/tensorflow/models/tree/master/research/deeplab
[2] https://ai.googleblog.com/2018/03/semantic-image-segmentation-with.html

pixel) e globale, allargando il filtro è possibile assimilare un contesto più elevato e ottenere risultati più soddisfacenti rispetto al riconoscimento sulla piccola porzione dell' immagine.
Con questo tipo di convoluzione è inoltre possibile mantenere inalterate le dimensioni della feature map. Con una standard convoluzione si renderebbe la feature map sempre più piccola, con possibile perdita di informazione con un significativo numero di livelli.

- **Atrous Spatial Pyramid Pooling** (ASPP): questa tecnica consiste nell' applicare in maniera parallela diverse Atrous Convolution sullo stesso input ma con rate diverse e successivamente fonderli insieme. Permette di migliorare l' accuratezza in quanto oggetti di classi uguali possono trovarsi scalati in maniera differente nell' immagine di input, permettendo un migliore riconoscimento in questo senso.

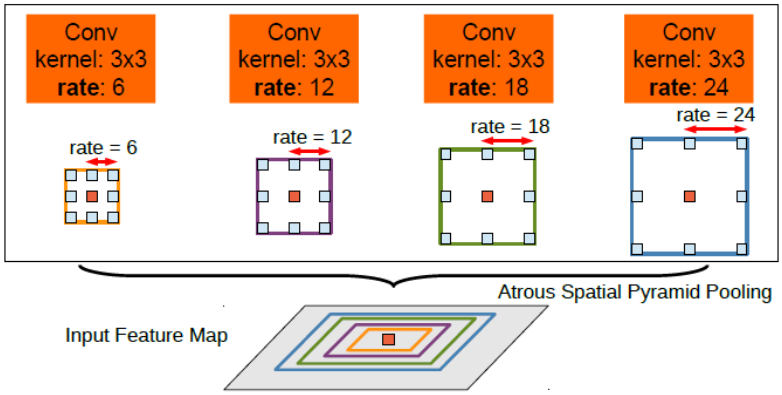

Figura 5: ASPP. Fonte: *towardsdatascience.com*

- **Decoder**: presente nella versione v3+, permette di rifinire i bordi della segmentazione. In questa fase è possibile controllare lo scalamento dell' immagine data come output.

Il workflow completo è visualizzabile in figura 6.

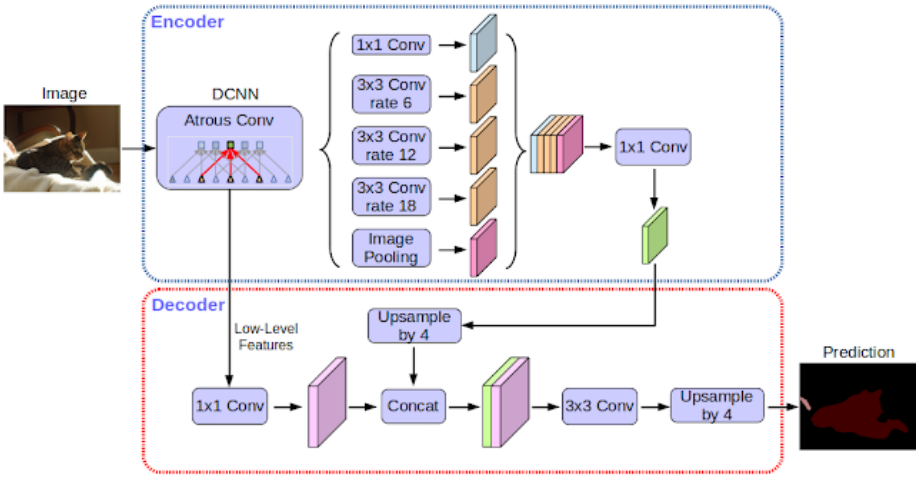

Figura 6: Workflow completo di DeepLabv3+. Fonte: Google AI blog

2.3 Implementazione della segmentazione semantica in una applicazione Android

Si è voluto creare una applicazione Android che sfruttasse TensorFlow Lite e DeepLab per eseguire della segmentazione semantica: l' obiettivo prefissato è quello di scontornare una figura di persona in modo da poter creare sticker per l' applicazione di messaggistica Telegram.
Come rete neurale è stata utilizzata DeepLabv3+ con backbone MobileNetv2 e allenata sul dataset ADE20K e successivamente convertita in modo tale da essere compatibile con TensorFlow Lite. Dopo variati esperimenti si è deciso di non ottimizzare la rete (sotto il punto di vista della quantizzazione) poichè si andava a perdere una decina di percentuali in accuratezza di fronte a pochi millisecondi di tempo di inferenza. In particolare gli sviluppatori promettono una accuratezza del 75%, comunque ottima data la piattaforma. La rete accetta come input immagini da 257x257 pixel, riconoscendo figure divise in ventuno classi differenti.

La rete restituisce una immagine dove per ogni pixel si ha uno score per ogni classe riconosciuta: è stato possibile evidenziare, come se fosse una maschera, la figura di una persona in quei pixel contenti score elevati per la classe 'person'. Dopo l' acquisizione della maschera si sono resi necessari diversi passi di processo in modo da migliorare il risultato finale:

- Riconoscimento della componente connessa più estesa: in condizioni sfavorevoli, la rete riconosce oltre alla persona, piccole aree non connesse e da rimuovere, come volti di persone in lontananza, sedili o semplicemente aree rosa sullo sfondo. É stato im-

plementato l' algoritmo di Hoshen–Kopelman per etichettare e isolare la componente connessa più estesa.

- Riempimento di eventuali buchi nella maschera: è capitato con persone dai lunghi capelli rossi che questi formassero dei buchi dentro la maschera, in quanto non riconosciuti dalla rete. Si è implementato un algoritmo di ricerca del bordo, rifinimento per connettività e successivamente un FloodFill per riempire completamente dall' interno la maschera, per poter migliorare il risultato. Gli algoritmi di ricerca del bordo sono stati anche utilizzati in quanto uno sticker per essere tale necessita di bordo bianco.

- Nascondere eventuale scostamento tra maschera e immagine: la rete restituisce nella maggior parte dei casi una maschera che è due-tre pixel più grande della figura. É stato risolto empiricamente posizionando un bordo di dimensione 2 pixel sulla maschera scalata 512x512.

Riferimenti bibliografici

[1] URL: https://developer.android.com/ndk/guides/neuralnetworks/.

[2] URL: https://www.tensorflow.org/lite/guide.

[3] URL: https://www.tensorflow.org/lite/convert/index.

[4] URL: https://google.github.io/flatbuffers/.

[5] URL: https://www.tensorflow.org/lite/guide/inference.

[6] URL: https://www.tensorflow.org/lite/performance/post_training_quantization.

[7] Liang-Chieh Chen e Yukun Zhu. «Semantic Image Segmentation with DeepLab in TensorFlow». In: (). URL: https://ai.googleblog.com/2018/03/semantic-image-segmentation-with.html.

[8] Liang-Chieh Chen et al. «Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation». In: *ECCV*. 2018.

[9] Loris Nanni. «Dispensa di Fondamenti di Intelligenza Artificiale». In: ().

[10] Mark Sandler et al. «MobileNetV2: Inverted Residuals and Linear Bottlenecks». In: *CVPR*. 2018.

[11] Sik-Ho Tsang. «Review: DeepLabv1 DeepLabv2—Atrous Convolution (Semantic Segmentation)». In: (). URL: https://towardsdatascience.com/review-deeplabv1-deeplabv2-atrous-convolution-semantic-segmentation-b51c5fbde92d.

[12] Sik-Ho Tsang. «Review: DeepLabv3—Atrous Convolution (Semantic Segmentation)». In: (). URL: https://towardsdatascience.com/review-deeplabv3-atrous-convolution-semantic-segmentation-6d818bfd1d74.